Постановка задачи:

1) Написать код, решающий антагонистическую матричную игру путем сведения ее к паре двойственных задач линейного программирования

2) Проиллюстрировать работу данного кода путем визуализации спектров оптимальных стратегий с помощью Jupiter Notebook для игр с:
   i. полным спектром оптимальных стратегий
   ii. неполным спектром оптимальных стратегий
   iii. спектром, состоящим из одной точки

Теоретическая справка:

Операция – совокупность целенаправленных действий

Контролируемые факторы – влияющие на исход операции величины

Стратегия – правило выбора контролируемых факторов

Пусть исход операции определяется стратегией оперирующей стороны $\mathrm{x} \in \mathrm{X}$ и стратегией другого субъекта $\mathrm{y} \in \mathrm{Y}$ (X,Y – множества стратегий сторон). Цель оперирующей стороны состоит в максимизации своей функции выигрыша $F(\mathrm{x},\mathrm{y})$, а второго участника – в максимизации его функции выигрыша $G(\mathrm{x},\mathrm{y})$. Такого рода конфликтная ситуация называется *игрой* двух лиц – игроков.

Антагонистическая игра: $G(\mathrm{x},\mathrm{y}) \equiv -F(\mathrm{x},\mathrm{y})$. Противник выбирает свою стратегию, зная стратегию оперирующей стороны.

Таким образом, антагонистическая игра задается набором

$$\Gamma = \langle X, Y, F(\mathrm{x},\mathrm{y}) \rangle$$

Пара $(\mathrm{x}^0,\mathrm{y}^0) \in \mathrm{X} \times Y$ называется *седловой точкой* функции $F(\mathrm{x},\mathrm{y})$ на $\mathrm{X} \times Y$, если

$$F(x,y^0) \le F(\mathrm{x}^0,\mathrm{y}^0) \le F(\mathrm{x}^0,\mathrm{y}) \text{ для любых } \mathrm{x} \in \mathrm{X} \text{ и } \mathrm{y} \in \mathrm{Y}.$$

Антагонистическая игра Г имеет *решение*, если функция $F(\mathrm{x},\mathrm{y})$ имеет на $\mathrm{X} \times Y$ седловую точку.

Пусть $(\mathrm{x}^0,\mathrm{y}^0)$ – седловая точка, тогда тройка $(\mathrm{x}^0,\mathrm{y}^0, v = F(\mathrm{x}^0,\mathrm{y}^0))$ называется *решением* игры, $\mathrm{x}^0,\mathrm{y}^0$ – *оптимальными* стратегиями игроков, а v – *значением* игры.

Антагонистическая игра Г называется *матричной*, если множества стратегий игроков конечны: $X = \{1,\dots,m\}$, $Y = \{1,\dots,n\}$. Стратегия 1-го игрока обозначается через i, а второго через j, а выигрыш первого $F(i,\mathrm{j})$ через a_{ij}. Матрица $\mathrm{A} = (a_{ij})_{m\times n}$ называется матрицей игры.

Тогда $(\mathrm{i}^0,\mathrm{j}^0)$ – седловая точка матрица A, если $a_{\mathrm{i}^0\mathrm{j}^0} \le a_{\mathrm{i}^0\mathrm{j}^0} \le a_{\mathrm{i}^0\mathrm{j}}$, $i = 1,\dots,m \;\; j = 1,\dots,n$

Иначе, элемент $a_{\mathrm{i}^0\mathrm{j}^0}$ является минимальным в i^0-й строке и максимальным в j^0-м столбце.

Рассмотрим игру Г с точки зрения первого игрока. Пусть он выбрал стратегию x, тогда его выигрыш будет не меньше, чем

$$\inf_{y \epsilon Y} F(\mathrm{x}, \mathrm{y})$$

Эта величина – *гарантированный результат (выигрыш)* для первого игрока, если он использует стратегию x. Наилучший гарантированный результат для первого игрока

$$\underline{v} = \sup_{x \epsilon X} \inf_{y \epsilon Y} F(\mathrm{x}, \mathrm{y})$$

называется *нижним значением* игры.

Определение. Стратегия x^0 первого игрока называется *максиминной*, если

$$\underline{v} = \inf_{y \epsilon Y} F(\mathrm{x}^0, \mathrm{y})$$

Рассмотрим игру Г с точки зрения второго игрока. Пусть он выбрал стратегию y, тогда для него гарантированным результатом будет величина

$$\sup_{x \epsilon X} F(\mathrm{x}, \mathrm{y})$$

Проигрыш второго будет не больше, чем эта величина.

Наилучший гарантированный результат для второго игрока

$$\bar{v} = \inf_{y \epsilon Y} \sup_{x \epsilon X} F(\mathrm{x}, \mathrm{y})$$

называется *верхним значением* игры.

Определение. Стратегия y^0 первого игрока называется *минимаксной*, если

$$\bar{v} = \sup_{x \epsilon X} F(\mathrm{x}, \mathrm{y}^0)$$

Необходимое и достаточное условие существование седловой точки:

1) Для того, чтобы функция $F(\mathrm{x}, \mathrm{y})$ на $\mathrm{X} \times Y$ имела седловую точку, необходимо и достаточно, чтобы было выполнено равенство

$$\max_{x \epsilon X} \inf_{y \epsilon Y} F(\mathrm{x}, \mathrm{y}) = \min_{y \epsilon Y} \sup_{x \epsilon X} F(\mathrm{x}, \mathrm{y}) \quad (1)$$

2) Пусть выполнено равенство (1). Пара $(\mathrm{x}^0, \mathrm{y}^0)$ тогда и только тогда является седловой точкой, когда x^0 – максиминная, а y^0 – минимаксная стратегии игроков.

Определение. *Смешанной* стратегией первого игрока в игре Г называется вероятностное распределение φ на множестве стратегий X

Пусть $X = \{1, \dots, m\}$, то есть игра матричная, тогда вместо φ смешанная стратегия будет обозначаться «вероятностным» вектором $p = (p_1, \dots, p_m)$, удовлетворяющим ограничениям

$$\sum_{i=1}^{m} p_i = 1, \qquad p_i \geq 0, \qquad i = 1, \dots, m$$

Если применяется вектор p, то стратегия i выбирается с вероятностью p_i.

Множество X – множество *чистых* стратегий.

Пусть

$P = \{ p = (p_1, \dots, p_m) \mid \sum_{i=1}^{m} p_i = 1, p_i \geq 0,\ i = 1, \dots, m \}$ - множество смешанных стратегий первого игрока,

$Q = \left\{ q = (q_1, \dots, q_n) \,\middle|\, \sum_{j=1}^{n} q_j = 1, \;\; q_j \geq 0, \;\; j = 1, \dots, n \right\}$ - множество смешанных стратегий второго игрока,

тогда математическое ожидание выигрыша первого игрока –

$$A(p, q) = \sum_{i=1}^{m} \sum_{j=1}^{n} p_i a_{ij} q_j$$

Таким образом, $\bar{\Gamma} = \langle P, Q, A(\mathrm{p}, \mathrm{q}) \rangle$ – смешанное расширение матричной игры Г.

! Всякая матричная игра имеет решение в смешанных стратегиях.

Основные свойства матричных игр со смешанной стратегией:

- Для игры с матрицей A справедливы следующие утверждения:
  1) $\min\limits_{q \epsilon Q} A(\mathrm{p}, \mathrm{q}) = \min\limits_{1 \leq j \leq n} A(\mathrm{p}, \mathrm{j})$ для любой стратегии $p \epsilon P$;
  2) $\max\limits_{p \epsilon P} A(\mathrm{p}, \mathrm{q}) = \max\limits_{1 \leq i \leq m} A(\mathrm{i}, \mathrm{q})$ для любой стратегии $q \epsilon Q$.

- Значение v игры с матрицей A может быть представлено в виде следующих двух формул:

$$v = \max_{p \epsilon P} \min_{1 \leq j \leq n} A(\mathrm{p}, \mathrm{j}) = \min_{q \epsilon Q} \max_{1 \leq i \leq m} A(\mathrm{i}, \mathrm{q})$$

- Для того, чтобы тройка $(\mathrm{x}^0, \mathrm{y}^0, v)$ была решением в смешанных стратегиях игры с матрицей A, необходимо и достаточно, чтобы

$$A(i, q^0) \leq v \leq \mathrm{A}(p^0, j), \;\; i = 1, \dots, m, \;\; j = 1, \dots, n,$$

что сводится к подсчету скалярных произведений

вектора p^0 на столбцы A: $\mathrm{A}(p^0, j) = \sum_{i=1}^{m} p_i^0 a_{ij}$

и вектора q^0 на строки A: $\mathrm{A}(i, q^0) = \sum_{j=1}^{n} a_{ij} q_j^0$

Сведение решения матричной игры к паре двойственных задач линейного программирования:

Значение матричной игры представимо в виде

$$v = \max_{p \epsilon P} \min_{1 \leq j \leq n} A(\mathrm{p}, \mathrm{j}) = \max_{p \epsilon P} \min_{1 \leq j \leq n} \sum_{i=1}^{m} p_i a_{ij}.$$

Введем вспомогательную переменную u и запишем задачу нахождения максимина как задачу ЛП

$$v = \max_{(u,p) \epsilon B} u \,,$$

где

$$B = \left\{ (u, p) \middle| \sum_{i=1}^{m} p_i a_{ij} \geq u, \;\; j = 1, \dots, n, \;\; \sum_{i=1}^{m} p_i = 1, \;\; p_i \geq 0, \;\; i = 1, \dots, m \right\}.$$

Это возможно, так как при фиксированном $p \epsilon P$ максимальное значение u при ограничениях $(u, p) \epsilon B$ равно $\min\limits_{1 \leq j \leq n} A(\mathrm{p}, \mathrm{j})$.

Сделаем замену переменных $z_i = \frac{p_i}{u}, z = (z_1, \dots, z_m)$ и учитывая ограничения $(u,p)\epsilon B$, получим

$$\sum_{i=1}^{m} z_i = \frac{1}{u},\ \ \sum_{i=1}^{m} a_{ij} z_i \geq 1,\ \ j = 1, \dots, n,\ \ z_i \geq 0,\ \ i = 1, \dots, m.$$

Отсюда

$v = \max\limits_{(u,p)\epsilon B} u = 1/\sum_{i=1}^{m} z_i^0$, где z^0 – оптимальное решение задачи ЛП:

$$\sum_{i=1}^{m} z_i \to min$$

$$\sum_{i=1}^{m} a_{ij} z_i \geq 1,\ \ j = 1, \dots, n,\ \ z_i \geq 0,\ \ i = 1, \dots, m.$$

По z^0 находим значение игры и оптимальную смешанную стратегию первого игрока:

$$v = 1/\sum_{i=1}^{m} z_i^0,\ \mathrm{p}^0 = vz^0$$

Аналогично получаем, что

$$v = \min_{q\epsilon Q} \max_{1 \leq i \leq m} A(\mathrm{i}, \mathrm{q}) = 1/\sum_{j=1}^{n} w_j^0,$$

где w^0 – оптимальное решение задачи ЛП:

$$\sum_{j=1}^{n} w_j \to max$$

$$\sum_{j=1}^{n} a_{ij} w_j \leq 1,\ \ i = 1, \dots, m,\ \ w_j \geq 0,\ \ j = 1, \dots, n.$$

Здесь $q^0 = vw^0 -$ оптимальная смешанная стратегия второго игрока.

Построенные задачи двойственны одна по отношению к другой.

Использование ЛП для решения матричной игры – наиболее эффективный прием, позволяющий использовать алгоритм симплекс-метода.

Симплекс-метод

Данный метод является методом целенаправленного перебора решений задачи линейного программирования. Он позволяет за конечное число шагов либо найти оптимальное решение, либо установить, что оптимальное решение отсутствует.

Исходная задача ЛП в стандартной форме:

$$Z = \sum_{j=1}^{n} c_j x_j \to max$$

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i,\ \ i = 1, \dots, m,\ \ x_j \geq 0,\ \ j = 1, \dots, n.$$

Алгоритм симплекс-метода:

I итерация:

1 этап: формирование исходной симплекс-таблицы.

Исходная задача линейного программирования задана в стандартной форме. Приведем ее к каноническому виду путем введения в каждое из ограничений-неравенств дополнительной неотрицательной переменной (*слабой* переменной) $x_{n+i}, i = 1, \dots, m$. Получим в результате:

$$Z = \sum_{j=1}^{n} c_j x_j \to max$$

$$\sum_{j=1}^{n} a_{ij} x_j + x_{n+i} = b_i, \;\; i = 1, \dots, m, \;\; x_j \geq 0, \;\; j = 1, \dots, n+m.$$

В полученной системе уравнений примем в качестве разрешенных (базисных) слабые переменные $x_{n+i}, i = 1, \dots, m$, тогда свободными переменными будут $x_j, \;\; j = 1, \dots, n$. Выразим базисные переменные через свободные:

$$x_{n+i} = b_i - \sum_{j=1}^{n} a_{ij} x_j \,, i = 1, \dots, m, \;\; x_j \geq 0, \;\; j = 1, \dots, n+m.$$

Приведем Z к следующему виду: $Z = 0 - (\sum_{j=1}^{n} -c_j x_j) \to max$

На основе полученной задачи сформируем исходную симплекс-таблицу:

	b	x_1	…	x_n	Оценочные отношения
x_{n+1}	b_1	a_{11}		a_{1n}	b_1/a_{1j}
…					
x_{n+m}	b_m	a_{m1}		a_{mn}	b_m/a_{mj}
Z_{max}	0	$-c_1$		$-c_n$	

Табл.1

2 этап: определение базисного решения.

Согласно определению базисного решения, свободные переменные равны нулю, а значения базисных переменных – соответствующим значениям свободных чисел, т.е.:

$$X = (0, \dots, 0, x_{n+1}, \dots, x_{n+m})$$

3 этап: проверка совместности системы ограничений ЗЛП.

Признак несовместности системы ограничений (признак 1):

Если нет строки с отрицательным свободным числом (кроме строки целевой функции), в которой не было бы хотя бы одного отрицательного элемента (т.е. отрицательного коэффициента при свободной переменной), то несовместности нет.

4 этап: проверка ограниченности целевой функции.

Признак неограниченности целевой функции (признак 2):

Если нет колонки с отрицательным элементом в строке целевой функции (кроме колонки свободных чисел), в которой не было бы хотя бы одного положительного элемента, то неограниченности нет.

5 этап: проверка допустимости найденного базисного решения.

Признак допустимости найденного базисного решения (признак 3):

Если найденное базисное решение не содержит отрицательных компонент, то оно является допустимым.

6 этап: проверка оптимальности.

Признак оптимальности (признак 4):

Если в строке целевой функции нет отрицательных элементов (свободное число данной строки при рассмотрении данного признака не учитывается), то решение оптимальное. Если решение неоптимальное, переходим к этапу 8.

7 этап: проверка альтернативности решения.

Найденное решение является единственным, если в строке целевой функции нет нулевых элементов (свободное число данной строки при рассмотрении данного признака не учитывается).

8 этап: определение разрешающего элемента.

8.1. Определение разрешающей колонки.

Так как найденное базисное решение допустимое, то поиск разрешающей колонки будем производить по следующей схеме: определяем колонки с отрицательными элементами в строке целевой функции (кроме колонки свободных чисел). Из таких колонок выбирается та, которая содержит наименьший элемент в строке целевой функции. Она и будет разрешающей. Пусть будет j-я колонка.

8.2. Определение разрешающей строки.

Для определения разрешающей строки находим положительные оценочные отношения свободных чисел к элементам разрешающей колонки, строка, которой соответствует наименьшее положительное оценочное отношение, принимается в качестве разрешенной. Пусть будет i-я строка.

Элемент, расположенный на пересечение разрешающей колонки и разрешающей строки, принимается в качестве разрешающего, то есть это будет элемент a_{ij}.

9 этап: преобразование симплекс-таблицы.

Разрешающий элемент показывает одну базисную и одну свободную переменные, которые необходимо поменять местами в симплекс-таблице, для перехода к новому «улучшенному» базисному решению. Переменные x_{n+i} и x_j в новой симплекс-таблице меняем местами.

9.1. Преобразование разрешающего элемента.

Разрешающий элемент преобразовывается следующим образом: $(a_{ij})^{-1}$

Полученный результат вписываем в аналогичную клетку новой таблицы.

9.2. Преобразование разрешающей строки.

Элементы разрешающей строки делим на разрешающий элемент данной симплекс-таблицы, результаты вписываются в аналогичные ячейки новой симплекс-таблицы.

9.3. Преобразование разрешающей колонки.

Элементы разрешающей колонки делим на разрешающий элемент данной симплекс-таблицы, а результат берется с обратным знаком. Полученные результаты вписываются в аналогичные ячейки новой симплекс-таблицы.

9.4. Преобразование остальных элементов симплекс-таблицы.

Преобразование остальных элементов симплекс-таблицы (т.е. элементов, не расположенных в разрешающей строке и разрешающей колонке) осуществляется по правилу «прямоугольника».

В исходной таблице мысленно вычерчиваем прямоугольник, одна вершина которого располагается в клетке, значение которой преобразуем, а другая (диагональная вершина) – в клетке с разрешающим элементом. Две другие вершины (второй диагонали) определяются однозначно. Тогда преобразованное значение клетки будет равно прежнему значению данной клетки минус дробь, в знаменателе которой разрешающий элемент, а в числителе произведение двух других неиспользованных вершин.

В результате данных преобразований получили новую симплекс- таблицу:

	b	x_1	…	x_{n+i}	…	x_n	Оценочные отношения
x_{n+1}	$b_1 \quad - \frac{a_{1j} * b_i}{a_{ij}}$	$a_{11} \quad - \frac{a_{1j} * a_{i1}}{a_{ij}}$	…	$-(a_{1j}/a_{ij})$	…	$a_{1n} \quad - \frac{a_{1j} * a_{in}}{a_{ij}}$	
…	…	…	…	…	…	…	…
x_j	b_i/a_{ij}	a_{i1}/a_{ij}	…	$(a_{ij})^{-1}$	…	a_{in}/a_{ij}	
…	…	…	…	…	…	…	…
x_{n+m}	$b_m \quad - \frac{a_{mj} * b_i}{a_{ij}}$	$a_{m1} \quad - \frac{a_{mj} * a_{i1}}{a_{ij}}$	…	$-(a_{mj}/a_{ij})$	…	$a_{mn} \quad - \frac{a_{mj} * a_{in}}{a_{ij}}$	
Z_{max}	$0 - \frac{-c_j * b_i}{a_{ij}}$	$-c_1 \frac{-c_j * a_{i1}}{a_{ij}}$	…	$-(-c_j/a_{ij})$	…	$-c_n \frac{-c_j * a_{in}}{a_{ij}}$	

Табл.2

В следующей итерации все этапы проводятся уже с новой таблицей.

Алгоритм останавливает свою работу, когда:

1. $a_{ij} \leq 0,\ i = 1, \dots, m,\ c_j > 0$, тогда функция неограниченно возрастает, то есть оптимального решения не существует;
2. $c_j \leq 0,\ j = 1, \dots, n$, тогда решение является оптимальным.

Библиотеки

- ***SciPy*** — библиотека с открытым исходным кодом, предназначенная для выполнения научных и инженерных расчётов. Она необходима для функции ***linprog*** из пакета ***scipy***.***optimize***, который содержит в себе множество алгоритмов оптимизации. Цель функции ***linprog*** - минимизация линейной объективной функции с линейными ограничениями равенства и неравенства. В нашей задаче для облегчения записи ***scipy***.***optimize***.***linprog*** будет обозначаться просто как ***linprog***.
- ***NumPy*** — библиотека с открытым исходным с такими возможностями, как поддержка многомерных массивов (включая матрицы), поддержка высокоуровневых математических функций, предназначенных для работы с многомерными массивами. В нашей задаче для облегчения записи ***NumPy*** будет обозначаться как ***np***.
- ***Matplotlib*** - визуализация решений. В нашей задаче для облегчения записи ***Matplotlib*** будет обозначаться как ***plt***.
- ***UnitTest*** - тестирование функций
- ***Sys***, ***Os*** – для подготовки пакетов (создания относительных ссылок)

Комментарии по работе *scipy*.*optimize*.*linprog*

Функция ***linprog*** из библиотеки ***SciPy*** необходима для решения двойственной задачи линейного программирования с помощью симплекс-метода.
Макет функции ***linprog*** выглядит так:

scipy.***optimize***.***linprog***(***c***, ***A_ub***=***None***, ***b_ub***=***None***, ***A_eq***=***None***, ***b_eq***=***None***, ***bounds***=***None***, ***method***=***'simplex'***, ***callback***=***None***, ***options***=***None***)

Она решает следующую задачу линейного программирования с матрицей A:

Минимизация:

$c^T * x \rightarrow min$

Для системы ограничений: $\begin{cases} A\_ub * x \leq b\_ub \\ A\_eq * x == b\_eq \\ x \geq 0 \end{cases}$

Рассмотрим подробнее параметры функции:

- ***c*** - Вектор коэффициентов для функции, которую необходимо минимизировать.
- ***A_ub***, ***A_eq*** - Матрицы с коэффициентами для системы неравенств, равенств соответственно.
- ***b_ub***, ***b_eq*** - Векторы правой части системы неравенств, равенств соответственно.

В рамках решения нашей задачи мы пользуемся только неравенством, поэтому используем лишь пару ***A_ub***, ***b_ub***.

- ***bounds*** : Определяет границы для переменных x_i. Не используется в задаче.
- ***method*** : '***simplex***' Тип метода, используемый для решения задачи. В данном случае мы используем симплексный метод.
- ***callback*** : Функция обратного вызова. Она может быть вызвана для каждой итерации симплексного алгоритма, например, для получения вектора решений для текущей итерации. В данной программе эта функция не используется.
- ***options*** : Опции. Можно задать максимальное количество итераций, разрешение печати сообщения о результате работы. Не используется.

Возвращаемым значением является объект ***scipy***.***optimize***.***OptimizeResult***, содержащий следующие интересующие нас поля:

- ***x*** – массив значений переменных, доставляющих минимум целевой функции.
- ***fun*** – возвращает значение целевой функции $c^T * x$. В случае единичного вектора ***c*** возвращается линейная форма ***x***.

Описание программы

На вход программе подается матрица выигрышей антагонистической игры.

Далее вызывается функция ***nash_equilibrium***(***a***, ***test*** = ***0***), где ***a*** – матрица выигрышей, ***test*** – переменная, отключающая визуализацию при значении ***1***. Ее значение по умолчанию ***0***.

Далее вызывается функция ***modetask***.***mini***(***a***) из пакета ***modetask***, созданного в рамках выполнения дополнительного задания.

Пакет ***modetask*** состоит из модулей ***minimum***, ***solution***, ***printall*** из которых согласно файлу ***__init__*** берутся функции ***mini***, ***sol***, ***allprint*** соответственно.

Чтобы пользоваться функциями этого пакета в итоговом файле, содержащем функцию ***nash_equilibrium*** мы его подключаем при помощи ***import modetask***. Чтобы это было возможно, мы меняем текущий каталог на родительский при помощи функции ***sys***.***path***.***append***(***os***.***path***.***join***(***sys***.***path***[***0***],***'..'***)). Эта функция необходима, потому как ***Python*** ищет пакеты по родительскому каталогу.

Вернемся к ходу работы: функция ***mini***(***a***) получается на вход матрицу выигрыша ***a***. Далее, локальной переменной ***minim*** присваивается значение ***a***.***min***(), т.е. значение минимального элемента матрицы. После этого матрица проверяется на неположительность путем сравнения ***minim*** с ***0*** условным оператором ***if***.

Если значение переменной > ***0***, то значение переменной меняется на ***0***, так как в этом случае матрица положительна и преобразовывать ее не нужно. Иначе (***minim ≤ 0***) ей присваивается значение ***1 +*** (-***1***)****minim***. После этих действий функция возвращает значение ***minim***.

В основной программе результат работы вызванной функции присваивается переменной ***minim***. Следующий шаг заключается в вызове функции ***sol*** из пользовательского пакета ***modetask***. На вход функции поступают уже известные нам матрица ***a*** и ***minim***.

Внутри функции ***sol*** матрица преобразуется в положительную. Для этого ко всем элементам матрицы прибавляется ***minim***. Если матрица была положительной, то с ней ничего не произойдет, потому что ***minim*** будет равна ***0***. В результате такого аффинного преобразования стратегии игроков не поменяются (в силу соответствующей теоремы), но поменяется значение цены игры, а любой элемент матрицы будет строго больше ***0***. Преобразование матрицы в положительную необходимо для унификации использования функции ***linprog***. Для матрицы с только положительными/отрицательными элементами в качестве вектора коэффициентов можно использовать вектора из ***1*** или -***1***.

Далее задаются две дополнительные матрицы, используемые для решения прямой и двойственной задач линейного программирования соответственно: ***matrix2*** = (-***1***)****np***.***transpose***(***a***); ***matrix1*** = ***a***. Функция ***transpose***(***a***) библиотеки ***numpy*** возвращает транспонированную матрицу выигрыша.

Следующий шаг — это решение задачи линейного программирования. Переменной ***result1*** присваивается решение прямой задачи, а ***result2*** – двойственной. Рассмотрим подробней решение этих задач:

- ***linprog***(***np***.***ones***(***len***(***matrix1***)), ***matrix2***, (-***1***)****np***.***ones***(***len***(***matrix2***))) – возвращает объект типа ***scipy***.***optimize***.***OptimizeResult***. Эта функция решает следующую задачу:

p → min

Для системы ограничений:

$$\begin{cases} -aT * p \leq (-1, \dots, -1), \\ p \geq 0 \end{cases}$$ которая переходит в $$\begin{cases} a * p \geq (1, \dots, 1) \\ p \geq 0 \end{cases}$$

scipy.***optimize***.***OptimizeResult***.***x – «***стратегия***»*** первого игрока.

scipy.***optimize***.***OptimizeResult***.***fun –*** линейная форма вектора-***«***стратегии***»***.

Отметим, что функция ***np***.***ones***(***n***) возвращает вектор из ***1*** длины ***n***, а функция ***len***(***vector***) возвращает длину заданного вектора. В случае двумерного вектора - матрицы ***len***(***a***)

возвращает число строк. Стоит заметить, что эта же функция для транспонированной матрицы возвращает число столбцов.

- Аналогично (***linprog***((-***1***)****np***.***ones***(***len***(***matrix2***)), ***matrix1***, ***np***.***ones***(***len***(***matrix1***)))) находится «стратегия» второго игрока. Отметим, что ***linprog***((-***1***)****np***.***ones***(***len***(***matrix2***)) преобразует задачу с нахождения минимума в нахождение максимума

Следующий шаг – нахождение цены игры, обозначаемой как ***price***. Для этого единицу делится на линейную форму «стратегии» первого игрока, т.е. ***price*** = ***1 / result***.***fun***

После этого находятся реальные стратегии первого и второго игроков путем домножения полученных ранее результатов на цену игры: ***strategy1***(***2***) = ***price * result1***(***2***).

На последним шагу мы вспоминаем, что искали цену игры для преобразованной матрицы. Поэтому для получения цены игры для исходной матрицы выигрышей, из полученной цены игры вычитается то число, на которое была сдвинута исходная матрица, т.е.
price = ***price*** – ***minim***

Таким образом, получены цену игры и стратегии первого и второго игроков. Они заносятся в единый вектор результата ***result***, который функция ***sol*** и возвращает:

Result = ***np***.***array***([***price***, ***strategy1***, ***strategy2***],***object***), где ***array*** – функция из библиотеки ***NumPy***, создающая вектор из заданных в квадратных скобках объектов. Опция ***object*** позволяет включать в вектор числовые переменные любых типов и размеров, т.е. задает вектор специального типа. Благодаря этому можно не беспокоится о размерности численных векторов, а также становится возможным сравнение таких объектов, правда, ограниченное проверкой равенства.

Получив в переменную ***result*** вектор-результат работы функции ***sol***, программа проводит проверку при помощи условного оператора ***if*** равенства параметра ***test*** нулю. Если он больше ***0***, то программа была запущена в режиме тестирования и визуализировать ее решение не нужно. В этом случае функция ***nash_equilibrium*** завершается и возвращает полученный вектор-результат. Иначе запускается функция ***allprint*** из пользовательского пакета.

На вход этой функции поступает все тот же вектор-результат. Функция печатает цену игры, стратегии первого и второго игроков с соответствующими комментариями. Для этого используется функция ***print***(****objects***, ***sep*** = ‘ ‘, ***end*** = ‘***n***’, ***file*** = ***std***.***stdout***), выводящая массив объектов ***objects*** (в данном случае это текстовый комментарий и элемент числового вектора) в файл ***file*** (по умолчанию это стандартный поток вывода ***std***.***stdout***), разделяя их строковыми объектами ***sep*** (по умолчанию это пробел) и ставя в конце строковый объект ***end*** (по умолчанию это символ конца строки ***n***).

Далее, используя функции из библиотеки ***MatPlotLib***, функция выводит двумерные графики с визуализацией стратегий первого и второго игроков. Рассмотрим, использованные для этого функции:

- ***plt***.***title*** (***txt***) – печать заголовка таблицы
- ***plt***.***xlabel*** (***txt***) – печать наименования оси абсцисс
- plt.***ylabel*** (***txt***) – печать наименования оси ординат
- ***plt***.***scatter***(***x***,***y***) – печать отдельных точек (***x***,***y***). Пары значений берутся последовательно из векторов ***x*** и ***y***, поэтому вектора должны быть одинаковой длины.

В имеющейся реализации задачи вектором ***x*** является вектор состоящий из последовательности натуральных чисел ***1***, ***2***, .. , ***1*** + ***len***(***result***[***i***]), которому ставится в соответствие вектор стратегии ***i***-го игрока.

- ***plt*.*grid***() – покзывает сетку графика
- ***plt*.*show***() – показывает сам график
- ***plt*.*figure***(***integer***) – выбор поля для печати фигуры. В программе графики для разных игроков печатаются в разных полях.

На этом работа основной программы завершается.

О *unit* - тестах

Модульное тестирование, или юнит-тестирование (англ. ***unit testing***)
— процесс впрограммировании, позволяющий проверить на корректность отдельные модул иисходного кода программы. Идея состоит в том, чтобы писать тесты для каждойнетривиал ьной функции или метода. Это позволяет достаточно быстропроверить, не привело ли очер едное изменение кода к регрессии, то есть кпоявлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Для их проведения используется библиотека ***UnitTest***. С ее помощью задается специальный класс тестов ***LinProgTest***(***UnitTest***.***TestCase***), состоящий из элементов ***setUp***, ***tearDown*** - функций, запускающихся в начале и конце каждого теста (в текущей версии они просто печатают соответствующие состояния: “***Set Up for test***”, “***Tear Down for test***”) и функций ***test0***,…,***testN***, в которых собственно и происходит проверка.

Test1 и ***Test2*** были использованы для сравнения результатов работы программы с данными, полученным эмпирическими. Они сверяют вектор стратегии первого игрока и цену игры, вычисленные вручную для матрицы ***2x2***, с решением, полученным в результате выполнения функции ***sol***. Для этого используется функция библиотеки ***UnitTest self*.*assertEqual***(***a***,***b***), которая в случае неравенства объектов ***a*** и ***b*** выдает ошибку.

О количестве ошибок и их видах можно узнать из потока вывода программы.